Software-Fehlersuche Reloaded

Die Zukunft der Debug-Technologien

André Schmitz, Green Hills Software
aschmitz@ghs.com

Debugging mittels "printf()", Setzen von Haltepunkten und Code-Ausführung via Einzelschritt sind die Debug Methoden von gestern. Heutige Debug-Tools können bereits viel mehr, wie zum Beispiel automatische Fehlersuche zur Laufzeit, Heap Analyse und Memory Leak Erkennung, Profiling und nicht zuletzt die Auswertung von Hardware Trace-Daten. Doch was wird in Zukunft kommen? Wie wird man komplexe Software auf Multi-Core Systemen, die massiv auf verschiedene Peripherie-Komponenten zugreift, schnell und einfach debuggen können? Wie wird in einem Team gemeinsame Fehlersuche betrieben werden? Welche Methoden werden uns bei der Arbeit helfen Zeit zu sparen? Dieses Papier zeigt den aktuellen Stand der Technik und gibt einen Ausblick auf neue Methoden und Technologien, die uns in Zukunft das Debuggen erleichtern werden.

Stand der Technik

Debugging ist heute schon mehr als single-stepping, breakpoints und Memory View. Schon heute gibt es Tools zur statischen Code Analyse oder MISRA-Checker, welche den Code sogar schon vor der ersten Ausführung analysieren und potentielle Fehler finden können. Dies ist die günstigste Art Software Fehler zu finden. Auch automatische Code Generierung ist ein großer Schritt hin zu fehlerfreiem Code.

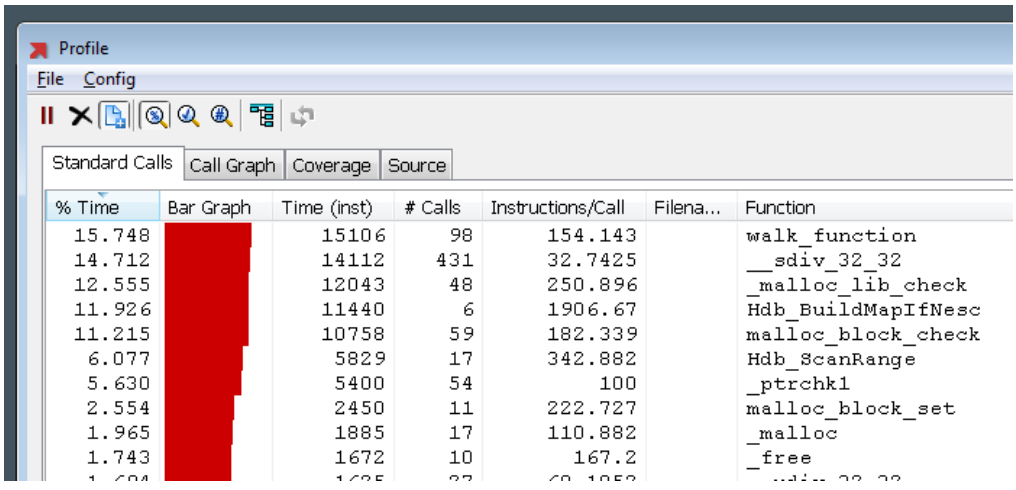

Abbildung 1: Ein Beispiel für Profiling Analyse

Wird das Programm auf dem Target ausgeführt, so kann man verschiedene Methoden nutzen Fehler im Programm leicht zu finden. Das Profiling erlaubt es zum Beispiel einfach die Hotspots des Programms zu identifizieren und zu entscheiden, an welchen Stellen im Programm man weiter optimieren sollte. Die Laufzeit-Fehlersuche erlaubt es, mithilfe des Compilers Überprüfungs-Sequenzen in den Code einzubauen, mit deren Hilfe man ganz automatisch Fehler finden kann. Diese Fehler sind zum Beispiel die Überschreitung von Array Grenzen, die Überschreitung des

Wertebereichs von Variablen, die Division durch Null oder die Dereferenzierung eines Null-Pointers.

Wenn es um die Verwendung von dynamischem Speicher geht, so kann man sich mit aktuellen Tools automatisch alle Reservierungen auf dem Heap anzeigen lassen, Zugriffe auf nicht allokierten Speicher identifizieren und nicht zuletzt automatisch alle Memory Leaks finden lassen. Es gibt Tools zur Analyse des zeitlichen Verhaltens eines Systems, in dem alle relevanten Ereignisse geloggt und visualisiert werden.

Ein weiterer wichtiger Aspekt der Fehlersuche ist die Simulation. Man kann Code zum Beispiel auf simulierten Controller ausführen oder das Verhalten der Sensorik und der Umwelt des Embedded Systems simulieren (Rest-Bus, Hardware in the Loop [1], etc.). Es gibt Werkzeuge mit deren Hilfe man automatisch Modul- und Integrationstests generieren kann um eine gute Testabdeckung und Test-Dokumentation zu erhalten.

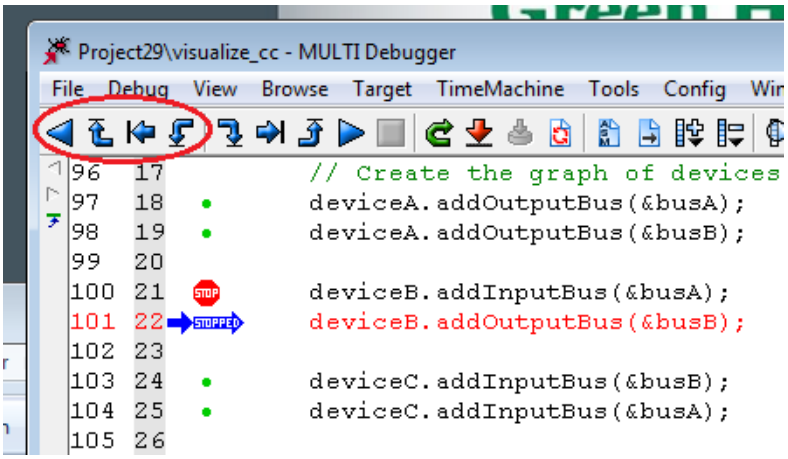

Abbildung 2: Zurück in der Zeit Debuggen

Viele Controller haben eine Trace Unit integriert, sodass man mit passenden Trace Tools die Trace Daten sammeln und sehr geschickt aufbereiten kann. Nicht zuletzt kann man mit entsprechenden Tools dann auch quasi rückwärts in der Zeit Debuggen, d.h. man wandert durch die Trace Daten indem man den Program-Counter im Debugger vor und zurück wandern lässt und dabei wie gewohnt Variablen und Register anzeigt.

Herausforderungen der Zukunft

Wenn wir nun weiter in die Zukunft schauen, dann müssen wir uns auf immer mehr Multi-Core CPUs mit immer höheren Takt-Frequenzen gefasst machen. Aktuelle High-End CPUs kommen mit 4 oder 8 Cores und einem Takt von mehr als 2GHz daher [2] [3]. Dieser Trend wird sicherlich weiter gehen, und dabei ist es schon fast egal was schneller wächst, die Zahl der Cores oder die Frequenz.

Doch wie bekommt man die Trace Daten von einer 8 Core 2GHz CPU auf eine Trace Hardware? Werden wir Tools haben, die uns die Analyse von nebenläufigen Anwendungen auf 8 oder mehr Cores über einen langen Zeitraum erlauben?

Es ist absehbar, dass wir mehr und mehr heterogene Hardware und Software haben werden. Hier will man gleichzeitig verschiedene Betriebssysteme auf einem Hypervisor oder verschiedenen Cores der gleichen Architektur (Cortex-M4 und Cortex-A9) oder sogar Kombinationen von ganz unterschiedlichen Hardware-Architekturen debuggen können.

Ein weiteres interessantes Problem wird die Analyse der Interaktion von Software und Hardware sein. Es gibt zum Beispiel für nVidia GPUs ein Tool mit Namen Nsight [4], welches die Abläufe auf der GPU und im Programm-Code beim rendern einer Szene darstellen kann und es erlaubt, diese effizient zu analysieren. Das Tool zeigt sehr gut die Interaktion zwischen CPU und GPU. Solche Tools gibt es derzeit aber nur für PCs, nicht aber für das den Embedded Bereich.

Ganz egal was wirklich kommen wird, wir werden auf jeden Fall in der Lage sein müssen mit immer größeren Datenmengen umzugehen und schnell und effizient zu durchsuchen, analysieren und visualisieren. Letztlich müssen wir uns auch fragen, wie wir schneller Softwarefehler finden können. Das erfordert zu allererst ein Umdenken und Optimierungen im Entwicklungsprozess.

Ausblick in die Zukunft

Lassen Sie mich den Ausblick in die Zukunft in Form von Fragen formulieren.

Wäre es nicht schön, wenn man große Datenmengen effizient visualisieren könnte, wo Sie zu jeder Zeit sehen können, was Ihr Programm wann genau macht oder gemacht hat? Dafür brauchen Sie natürlich schnelle Werkzeuge auf Ihren Arbeitsplatzrechnern. Das Tool Nsight von nVidia ist optimiert für die Analyse von PC Programmen, also Programmen, die auf demselben PC laufen, wie das Tool. Wäre es nicht schön so ähnliche Tools für ein Embedded System auch nutzen zu könnten? Wir könnten dann sehr komplexe Software Systeme so laufen lassen, dass man alle Vorgänge innerhalb und außerhalb des Systems in allen Details erkennen und korrelieren kann. Vielleicht benötigen wir dazu neue Simulationsumgebungen, wie wir sie heute noch nicht haben.

Sind wir schnell genug bei der Identifizierung der Ursache eines Software Fehlers? Wenn wir es schaffen unsere Prozesse und das Tooling so zu optimieren, dass die Zeit vom Auftreten und Erkennen eines Software Fehlers bis zum Finden und Beheben der defekten Code-Stelle immer kürzer wird, dann steigert das die Produktivität ungemein. Wir werden damit auch die immer komplexeren Software Systeme einfach debuggen können.

Wie sollen wir in Zukunft in immer größeren Software Teams arbeiten wenn immer komplexere Software immer obskurere Fehler hervorbringen? Wenn eine Tester einen schwer zu reproduzierenden Fehler gefunden hat, dann muss er dem Entwickler eine Umgebung zur Verfügung stellen können, in der dieser das Problem in kürzester Zeit wieder herstellen, sowie den Fehler identifizieren und damit beheben kann.

Zusammenfassung

Die Herausforderung bei der Software Fehlersuche werden sein

- Verarbeiten von großen Datenmengen
- Minimieren der Zeit vom Auftreten eines Fehlers bis zum Beheben desselben
- Effiziente Kooperation in Software Teams

Wir brauchen neue Konzepte, Technologien und Werkzeuge um die Produktivität der Software Entwickler zu steigern.

Referenzen

[1] https://de.wikipedia.org/wiki/Hardware_in_the_Loop
[2] https://de.wikipedia.org/wiki/Intel_Atom
[3] https://de.wikipedia.org/wiki/ARM_Cortex-A
[4] http://www.nvidia.com/object/nsight.html

Autor

Andre Schmitz erhielt sein Diplom in Physik 1997 an der Universität Bonn. Anschließend entwickelt er bei der FhG Steuerungs- und Simulations-Software für Autonome Roboter. Von 2000 bis 2005 entwickelte Herr Schmitz Embedded Software für UMTS Kommunikationssysteme. Seit 2005 ist Herr Schmitz bei Green Hills Software für die technische Unterstützung von Kunden und die Durchführung von Schulungen zuständig. Herr Schmitz ist seitdem regelmäßig Referent bei diversen Fachkonferenzen.